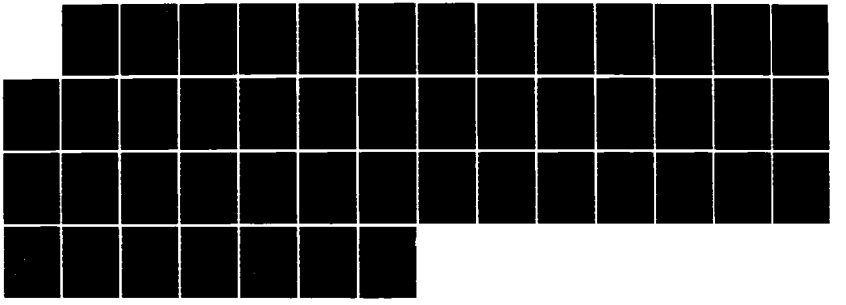
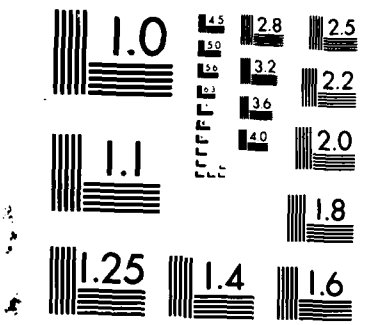
AD-A173 487 INITIAL COMPARISON OF MONOTONIC LOGICAL GRID AND ALTERNATIVE DATA BASE STRUCTURES(U) NAVAL RESEARCH LAB WASHINGTON DC J M PICONE ET AL. 30 SEP 86 NRL-MR-5860 1/1

UNCLASSIFIED F/G 9/2 NL





Naval Research Laboratory

Washington, DC 20375-5000 NRL Memorandum Report 5860 September 30, 1986

Initial Comparison of Monotonic Logical Grid and Alternative Data Base Structures

J. M. PICONE, S. G. LAMBRAKOS, AND J. P. BORIS

Laboratory for Computational Physics

S. JAJODIA

Computer Science and Systems Branch
Information Technology Division

AD-A173 487

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Memorandum Report 5860	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Research Laboratory 6b OFFICE SYMBOL (If applicable) Code 4040	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION Strategic Defense Initiative 8b OFFICE SYMBOL (If applicable) SDIO	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) Office of the Secretary of Defense Pentagon, Washington, DC	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 6323C PROJECT NO. SDI-0 TASK NO WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
Initial Comparison of Monotonic Logical Grid and Alternative Data Base Structures

12 PERSONAL AUTHOR(S)
Picone, J. M., Lambrakos, S. G., Boris, J. P., and Jajodia, S.

13a TYPE OF REPORT	13b TIME COVERED FROM ____ TO ____	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
		1986 September 30	47

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Monotonic Logical Grid, Dynamic Data Structures, Near Neighbors Problem

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

We compare the Monotonic Logical Grid (MLG) data base structure to alternative data base structures for tasks relevant to computing the dynamics of N moving objects. The tasks include identifying near neighbors of designated nodes, retrieving information on the near neighbors, and ordering this information according to distances of the associated near neighbors from the designated nodes. We use a collection of $N = 64$ K (65,536) noninteracting objects with randomly initialized velocities. We have performed the calculations on the Naval Research Laboratory (NRL) Cray X-MP computer, which has a hardware gather-scatter capability. We consider two types of alternative data structures for which the indexing is static (the data corresponding to each node always have the same index and memory locations). These two types encompass most of the data structures currently in use in manybody simulations. Data structure "Type 1" carries no additional information or "pointers" which could identify the near neighbors of each node. Data structure "Type 2" *does* carry coarse information on near neighbors by maintaining linked lists or a related system of "pointers" that change with the motion of nodes in time. Our numerical tests show that the MLG data structures are vastly superior to the

(Continued)

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL J. Michael Picone	22b TELEPHONE (Include Area Code) (202) 767-3055 22c OFFICE SYMBOL Code 4040

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Type 1 data structure when we require near-neighbor information on a large number M of designated or "focal" nodes. This occurs because the process of identifying near neighbors requires M identical sorting or partitioning processes per frame (or time step) in the case of Type 1 data base structures while only one sort per frame is necessary to maintain the ordering of data in the MLG data structure. We also find that the time for ordering nodes according to respective distances from the M focal nodes for Type 1 data structures is directly proportional to $M \times N$. Our tests show that the corresponding time in the case of the MLG is of order M. The MLG provides several advantages over Type 2 data structures, even though the respective operation counts are quite similar. The advantages include efficiency of memory allocation and memory management, smaller memory requirements, vectorizability and parallel partitioning, and simplicity of programming. The last property should lead to computer software with a reduced error rate and code which is more amenable to revision.

CONTENTS

INITIAL COMPARISON OF MONOTONIC LOGICAL GRID AND ALTERNATIVE DATA BASE STRUCTURES

1. INTRODUCTION

The Monotonic Logical Grid (MLG) algorithm organizes data associated with a collection of nodes so that spatial or geotemporal near neighbors will also be near neighbors in index space. Appendix I gives a simple example showing how an MLG might be constructed given a set of nodes with arbitrary spatial locations. Arranging data according to an MLG vastly reduces the work required for any calculation in which the near neighbors of each node must be identified or analyzed. One finds examples of such requirements in molecular dynamics (Hockney and Eastwood, 1981; Lambrakos and Boris, 1986) and sensor data analysis and target tracking and correlation (e.g., Reid, 1979 or Boris, Picone, and Lambrakos, 1986). The restructuring portion of the MLG algorithm, which maintains the proper ordering of memory, is of computational complexity $N \log N$, where N denotes the number of nodes contained in the data base. In addition, this iterated restructuring procedure is ideally suited for highly parallel and vector computers.

Here we compare the MLG data base structure directly with two other types of data base structures that might be used for the same tasks. For each of these alternative data base structures, the data associated with each node (e.g., spatial and velocity coordinates) remain in the same memory locations throughout the evolution of the system. We use the term "static" for this type of memory allocation. The first type of data base structure maintains no additional information or "pointers" identifying the near neighbors of each node. Such a "Type 1" data base might determine near neighbors of a given node by computing the N-1 distances to the other nodes and then ordering these distances according to a particular definition of *near* neighbors. Another method would be to select those nodes whose spatial coordinates fall within a prescribed interval about the coordinates of each designated or "focal" node. To illuminate the differences between this "Type 1" approach and the MLG, when implemented on a modern, high-speed computer, we have run a

Manuscript approved August 4, 1986.

series of tests on the NRL Cray X-MP with a system containing 64 K (65,536) nodes. As initial conditions we have chosen regular spacing on a cubical lattice with random initial velocities. This favors the MLG less than would a situation in which the nodes were all moving in roughly the same direction at roughly the same speeds. We ran the following tests on the MLG and a representative Type 1 data base structure:

(1) Find the indices of a specified number of near neighbors of given designated or "focal" nodes and write these to a buffer. We implement this in "Test 1" by computing the *N-1* distances of the nodes from each focal node and then ordering these distances from the smallest to the largest. To bracket our results, we choose two other, less comprehensive, and less expensive definitions of near neighbors for the Type 1 data structure. In "Test 1A" we identify those neighboring nodes which are within a prescribed distance of each focal node. In "Test 1B" we identify those nodes whose spatial coordinates fall within a prescribed interval about the spatial coordinates of each focal node. The above three definitions of "near neighbors" carry varying amounts of implicit information on the vicinities of the focal nodes.

(2) Write the data associated with the near neighbors identified in Test 1 to a buffer. This requires random sifting through the Type 1 (and Type 2) data base structures and involves inefficient short loops with the MLG, when implemented on the Cray X-MP.

(3) Order the data on the sets of near neighbors identified in Tests 1, 1A, and 1B according to distance of each near neighbor from a given focal node. Thus we identify the *nearest* neighbors of each focal node.

The next section presents the results of these tests and converts the data to equations for predicting the relative performance of the two approaches.

The second class of data structures which we consider as alternatives to the MLG are also static. However, these "Type 2" data structures maintain some dynamic information or "pointers" for identifying the near neighbors of each node. As an example, we will discuss the method of Hockney and Eastwood (1981), in which the physical space is divided into cells and linked lists are used to identify the particular nodes located in each cell at a

particular time. As a given node passes from one cell to a neighboring cell, the index of the node disappears from the list associated with the former cell and appears in the list corresponding to the cell that the node has just entered. Prior to the MLG, this was one of the best methods of computing near-neighbor interactions. The reader who is familiar with the MLG concept will recognize this alternative as being a distant relation of the MLG algorithm. Rather than swapping the data in memory to maintain the knowledge of spatial or geotemporal relationships among nodes, Hockney's Type 2 data base essentially swaps information (node indices) between linked lists. The linked lists associate data which have a fixed location in memory to fixed cells in space. These linked lists represent extra memory and a scalar component of the restructuring computation which is absent with the MLG. The price paid in the MLG data base structure is the need to move much more data than the linked lists require. We explore Type 2 data structures in Section 3.

Section 4 provides our conclusions concerning the data structures that we have considered. We then discuss the relationship of our results to the problem of tracking and correlation.

2. Monotonic Logical Grid vs. "Type 1" Data Base Structures

2.A. Description of Tests

The test problem which we have run on the Cray X-MP consists of 64 K noninteracting nodes moving randomly in a cubical volume. The boundary conditions in each x - y plane are skew-periodic (Lambrakos and Boris, 1986), and we used reflecting boundary conditions perpendicular to the x - y planes at each end of the z-domain. We prevent boundary conditions from influencing our results by choosing focal nodes which lie near the center of an x-y plane and which have a sufficient displacement from the lowest and highest z-values in the computational domain. Each node has a set of MLG indices and a single, constant Type 1 index or identification (ID) number. As the system evolves, the MLG indices of a node will change while the ID will not. We may thus interpret ID as a label of the static memory locations of the data on the nodes.

We have optimized the Type 1 data base operations so that the distances between a focal node at (x_f, y_f, z_f) and all others are computed in a fully vectorized fashion. For Test 1, we actually sort on the square of the internode distance

$$R_{id}^2 = (x_{id} - x_f)^2 + (y_{id} - y_f)^2 + (z_{id} - z_f)^2 \tag{1}$$

rather than the distance itself. This saves time by eliminating the additional square root operation required for computing the distance. In Eq. (1), the subscript f denotes the focal node, and id is the index of running over the other nodes in this system.

For Test 1 of the Type 1 data base, we use the HEAPSORT algorithm (Nijenhuis and Wilf, 1978) for the ordering according to R_{id}^2. We have chosen HEAPSORT because of the order N *log* N computational complexity and because the algorithm is a standard one in the theory of sorting. Our routine is not vectorized. For Test 1 the ordering of nodes by HEAPSORT accounts for most of the time expenditure, so that a significantly faster sorting algorithm would yield correspondingly reduced time costs for Test 1. Vectorizing on the

Cray X-MP typically yields an improvement in speed by a factor of 5 to 10. We mention that the restructuring algorithm required to maintain the MLG is also not vectorized in the present tests. The relative timings should therefore be meaningful even if the absolute computation rates might not. Notice that, for the Type 1 data base, Test 3 above ("find the *nearest* neighbors") is automatically satisfied upon completion of Test 1.

For Test 1A, we test the quantities R_{id}^2 against R^2, the square of a specified radius vector centered at each focal node. The values which we use are $R = 2\delta, 3\delta, 4\delta$, and 5δ, where δ is a constant that is approximately equal to the average difference in the x, y, or z coordinates of adjacent nodes. For Test 1B, we test the coordinates (x_{id}, y_{id}, z_{id}) to find the nodes whose locations fall within the interval from $(x_f - m\delta,\ y_f - m\delta,\ z_f - m\delta)$ to $(x_f + m\delta,\ y_f + m\delta,\ z_f + m\delta)$, where m is an integer. We have run the test with $m = 2$ and $m = 4$.

In the case of Type 1 data base structures, the retrieval of data associated with the near neighbors (Test 2) after the sorting is completed requires random sifting through the data base structure. For a particular focal node and time step, the IDs of the near neighbors will be random because of the random motion of the particles. This random sifting is called a "gather" operation, for which the NRL Cray X-MP has a special hardware capability. Thus, the Type 1 data structure can attain an appreciable fraction of vector speed when retrieving data associated with the near neighbors. This apparent gain relative to the MLG will be difficult to sustain in a fully parallel computing environment.

The time expenditure in using the MLG data structure depends on the following:

(1) How much swapping must be done at each time step or "frame" to maintain the indexing in MLG order.

(2) The number of "near neighbors" which must be accessed and processed for each focal node. Because of the MLG organization of computer memory (index space), we know that the near neighbors and the focal node will have a contiguous set of indices.

(3) The ability of a given computer system to capitalize on the fact that a contiguous set of indices identifies the data corresponding to near neighbors of a given node. Even on a conventional scalar machine, this provides an advantage, in that the near-neighbor data may be accessed through the use of DO-loops without performing a "gather" operation.

To be more specific, denote the indices of a focal node by the set of integers (i_f, j_f, k_f). To define the set of the near neighbors, we must specify the size of the index interval from $(i_f - \Delta i,\ j_f - \Delta j,\ k_f - \Delta k)$ to $(i_f + \Delta i,\ j_f + \Delta j,\ k_f + \Delta k)$, within which the indices of "near neighbors" will fall. That is, we require only the set of integers $(\Delta i, \Delta j, \Delta k)$, which we call the "maximum index offsets" of the set of near neighbors.

The appropriate maximum index offsets to use in retrieving data depend primarily on the number of *nearest* neighbors which the user wishes to identify. As shown in previous papers, the MLG provides coarse quantitative information on the identities of the nodes which are closest spatially to a focal node. This information is coarse because displacement in index space does not correspond perfectly to geometric or geotemporal displacement. Some of the nodes within a given index interval (as defined above) might actually be farther from the focal node than some of the nodes whose indices fall outside the interval. Thus one must be careful to retrieve data from a sufficiently large interval to include the desired number of *nearest* neighbors. Fortunately the cost of this safety factor is minor in practice, since the number of near neighbors which must be considered will only be a small fraction of the total number of nodes N, when N is large (Lambrakos and Boris, 1986).

Given the dependence of MLG data structure performance on (1) and (2) above, we have parameterized our tests on the basis of two quantities. The first is a dimensionless time interval between frames, given by

$$\mathrm{DT} = v_r \delta t / \delta s \ . \tag{2}$$

Here v_r is a characteristic maximum relative velocity of two neighboring nodes. For the present tests, v_r is the maximum value of a given velocity component (e.g., x-component)

which any node could have. In Eq. (2), δt is the value of the time interval between frames in the simulation, and δs is the minimum average distance between particles along the three coordinate axes. Expressed mathematically,

$$\delta s = min\{\langle \delta x \rangle, \langle \delta y \rangle, \langle \delta z \rangle\},$$

where

$$\langle \delta x \rangle = \frac{1}{(N_x - 1)N_y N_z} \sum_{k=1}^{N_z} \sum_{j=1}^{N_y} \sum_{i=1}^{N_x - 1} |x_{i+1\,jk} - x_{ijk}|$$

with similar expressions for $\langle \delta y \rangle$ and $\langle \delta z \rangle$. Thus, if the average internode distance along the x-axis is smaller than the similar quantities along the y- and z-axes, we choose $\delta s = \langle \delta x \rangle$. We define v_r and δs in this manner to determine a characteristic time interval over which internode crossings occur. Other definitions are possible and would merely result in a rescaling of DT. The quantity DT indicates the frequency with which nodes will pass each other during a time step or frame interval. This, in turn, measures how much work (sorting) is required to maintain (or restructure) the MLG after each framing interval. In molecular dynamics calculations, DT is usually less than one (Lambrakos and Boris, 1986). We have run tests with values of DT up to 3.0. For completeness, we mention that, in our tests, the maximum of any velocity component (e.g., x-component) was 4.0×10^5 cm/s and that the average displacement of nodes along any axis was 1.0×10^{-7} cm. Given this discussion, we assert without proof that any values of v_r, δt, and δs giving the same values of DT would yield the same conclusions as we reach below.

The second group of parameters which affect the results for the MLG data structure is the set of maximum MLG index offsets $(\Delta i, \Delta j, \Delta k)$ defining the vicinity of a given focal node from which we must retrieve the near neighbor data. That is, for a given set of MLG focal node indices (i_f, j_f, k_f), we will retrieve data for near neighbors whose three indices lie in the intervals:

$$\begin{aligned} i_f - \Delta i \le i \le i_f + \Delta i \\ j_f - \Delta j \le j \le j_f + \Delta j \\ k_f - \Delta k \le k \le k_f + \Delta k \end{aligned} \tag{3}$$

For larger offsets, we must compute more distances and require more sorting time to identify the *nearest* neighbors (Test 3). In our present tests, we have used offsets of 1, 2, 3, and 5.

In running the tests, we actually used five focal nodes to get an average time expenditure per focal node for each test. The variations in results for the focal nodes were negligible, so that averages based on five nodes are quite adequate. We ran each test calculation for 500 frames (time steps) and performed the tests at intervals of 100 frames to ensure that the ensemble had undergone significant changes due to the random velocities of the nodes. We then computed timings for the various tests in seconds per frame per focal node.

We point out that the MLG restructuring time per frame is independent of the number of focal nodes used, so that the time required *per focal node* is inversely proportional to the number of focal nodes processed. In molecular dynamics calculations or target correlation and tracking problems, an appreciable number of the nodes in the system must be processed as "focal nodes" at each time step or frame. Thus, the MLG restructuring time per frame per focal node will be quite small for realistic cases. In our test calculations, we did not vectorize the MLG restructuring algorithm, so that the MLG results represent upper bounds on the required sorting time. To perform Test 3 for the MLG, we had to compute distances of the near neighbors from the focal nodes and then render these in ascending order. To do so, we used the same HEAPSORT algorithm as that used in ordering the Type 1 data base. Obviously a vectorized sorting module would have done a faster job on this sort as well. In fact, the structure of the MLG itself contains enough information on nearest neighbors so that no sorting should be required at all. We conclude that our MLG timing results represent worst cases, again a conservative comparison.

2.B. Test Results

We present test results through the use of graphs and tables. Some procedural points are important to understanding the discussions. First, the cost of Test 1 (identifying *near*

neighbors) should include sorting the nodes to restructure the MLG in order to correspond to Tests 1, 1A, and 1B of the Type 1 data structure. The latter performs Test 1 (Test 1A) by sorting (partitioning) the nodes according to values of R_{id}^2 and performs Test 1B by partitioning nodes according to the values of the spatial coordinates. Our graphs of Test 1 costs includes the respective MLG and Type 1 sorting or partitioning times. However, in Tables 1 and 2 of MLG results, we have broken the sorting time out and written it in a separate column because the cost of maintaining the MLG is independent of the number of focal nodes in the problem. The remainder of the cost associated with Test 1 is proportional to the number of focal nodes, as is the time which the Type 1 data base spends sorting according to the various Test 1 criteria.

The second point concerns the procedure for finding *nearest* neighbors (Test 3). For Test 1 of the Type 1 data structure, we sort according to distance from a given focal node and incur no additional cost for determining *nearest* neighbors. The procedure for finding nearest neighbors in the case of the MLG involves two steps:

(1) Identify enough *near* neighbors to contain the desired number of *nearest* neighbors. The definition of "enough" will determine the maximum index offsets $(\Delta i, \Delta j, \Delta k)$ of near neighbors from each focal node. For the MLG, the maximum offset along each axis should be no more than five for most practical problems.

(2) Order the set of near neighbors according to internode distance R^2 to determine the required number of nearest neighbors.

This would also be the procedure for the Type 1 data base structure, if we used the method of Test 1A or 1B to find near neighbors. Step (2) is much less expensive than the HEAPSORT performed in Test 1 of the Type 1 structure when the total number of targets N is large. This is true because, in practice, the number of near neighbors identified in Step (1) is much smaller than N. A final point is that this determination of spatial *nearest* neighbors would *not* be performed when the MLG is in use. Rather, a direct parallel calculation of interactions involving the *near* neighbors in index space will most likely be

performed. Once again, this is because the *nearest* neighbors form a subset of those nodes already selected through index offsets as *near* neighbors.

2.B.1. Graphs of Costs versus Number of Focal Nodes

Figures 1 - 3 present the results of our tests on the MLG data base structure, and Figs. 4-6 present our results on the Type 1 data base structure. We express the costs of each test in seconds per frame and use a log-log scale in all cases. This is because our tests cover a wide range in number of designated or "focal" nodes, for which we are finding near neighbors. In most applications of interest, a large fraction (if not all) of the nodes will be processed as focal nodes. *The reader should exercise caution in comparing the graphs, as the maximum order of magnitude on the vertical scale will vary from figure to figure.*

Figure 1 shows the results of Test 1 (find a specified number of near neighbors and write the MLG indices and ID to a buffer) for the MLG data structure. As indicated above, the near neighbors are selected in groups defined by maximum index offsets from the indices of each focal node (Eq.(3)). If the maximum offsets for i, j, and k have the same value Δ, then the number of near neighbors is

$$N_{nn} = (2\Delta + 1)^3 - 1 \ . \tag{4}$$

As shown in Fig. 1, we used maximum offsets of 1, 3, and 5. In addition, we used a dimensionless time step DT = 0.6, which is reasonable for most problems. At small numbers of focal nodes, the cost goes asymptotically to the time required to maintain the MLG from frame to frame. This restructuring process depends on DT and N, but not on N_f, the number of focal nodes. We have included this "sorting" time with Test 1 as a natural part of the cost in determining near neighbors. The corresponding sorting time for the Type 1 data base constitutes most of the cost of Test 1 in that case. At higher values of N_f, the cost of the MLG near-neighbor access and write begins to increase linearly with N_f.

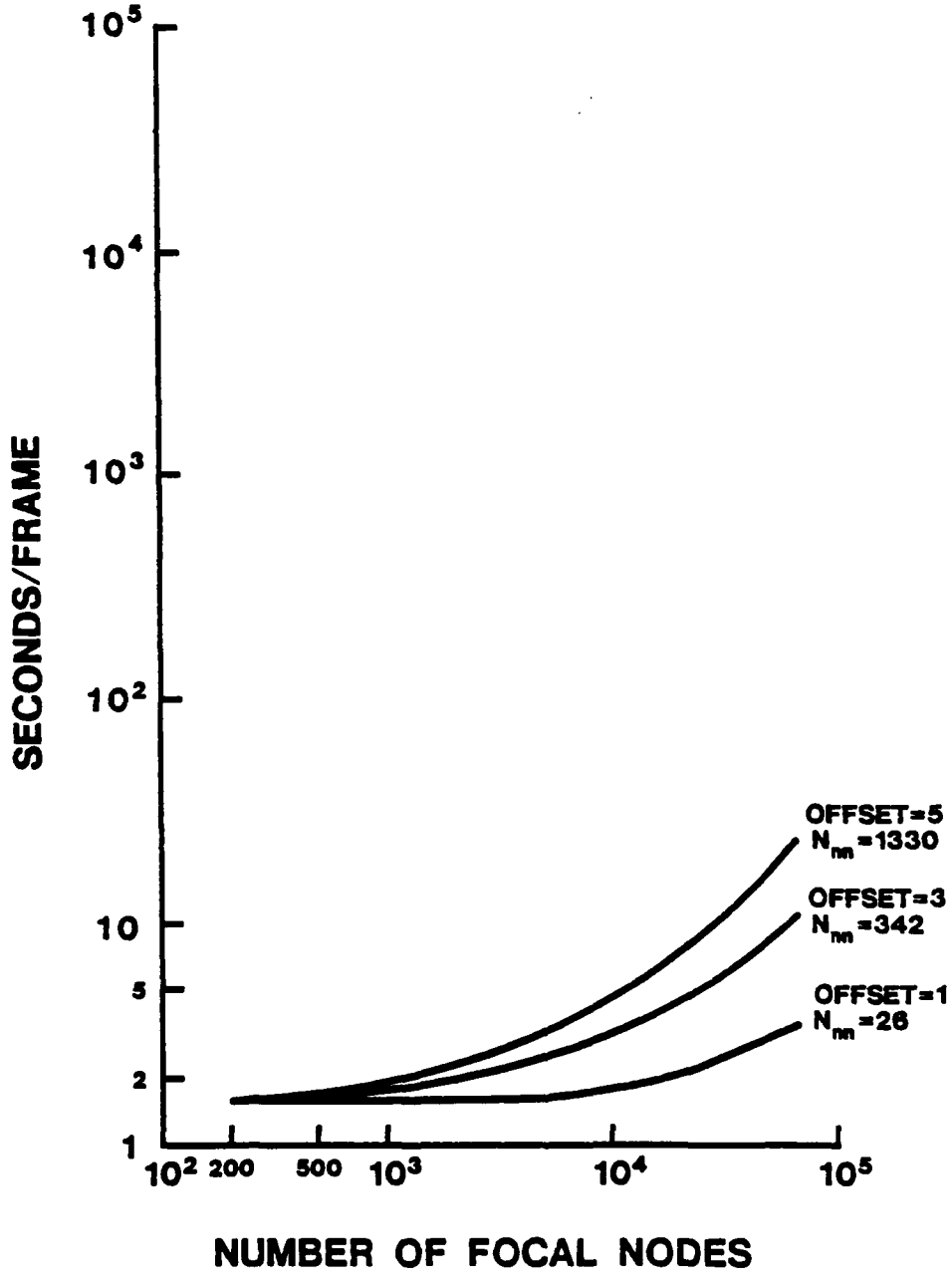


Fig. 1. Cost per frame of using a Monotonic Logical Grid data structure on the NRL Cray X-MP computer. Test 1: Identify near neighbors of each focal node. The total number of nodes is 64 K and the dimensionless time interval between frames is DT = 0.6.

Figure 2 shows that accessing (retrieving) the data associated with the near neighbors (MLG Test 2) varies linearly with N_f and becomes more expensive than the simple writing of near neighbors indices. This is primarily because we had a significant number of data arrays: three position coordinates, three velocity coordinates, and two extra words per particle for other information. Notice that the vertical scale of Fig. 2 is three orders of magnitude lower than the vertical scale of Fig. 1.

Figure 3 shows the cost of ordering the near neighbors according to distance from their respective focal nodes (MLG Test 3). This would be the final step in determining nearest neighbors using an MLG data structure. However, this is not really necessary with a highly parallel processor, since one could simply perform any "nearest-neighbor" interaction or correlation calculation by looping over the near neighbors identified in Test 1. We can see that at large N_f MLG Test 3 costs more than Tests 1 and 2. We again note that the cost curves depend primarily on the choice of maximum index offset Δ rather than the total number of nodes N. In addition, the cost scales as $N_{nn}\ log\ N_{nn}$ for a given value of N_f.

Figure 4 shows the cost of Test 1 for the Type 1 data structure. In order to find near neighbors, we have computed the distances of all nodes from a given focal node and then used a HEAPSORT routine to render the distances in ascending order. Thus we have actually computed nearest neighbors (Test 3) at the same time. Notice that this process depends on the total number of nodes N along with N_f; the scaling is $N\ log\ N$ because of the use of a HEAPSORT routine. We have used this scaling to plot the two dashed curves for smaller values of N. The curves terminate at $N_f = N$. A comparison of Figs. 1 and 4 shows that the cost of this near neighbors calculation for N = 64 K is more than three orders of magnitude more expensive than accessing near neighbors with an MLG.

Figures 3 and 4 show a difference of 2 - 3 orders of magnitude between the Type 1 data structure (Test 1) and the MLG in the cost of finding *nearest* neighbors when N = 64 K. This need not be true for significantly smaller values of N, if relatively large index offsets are needed in the MLG for an accurate calculation. Such would be the case if the

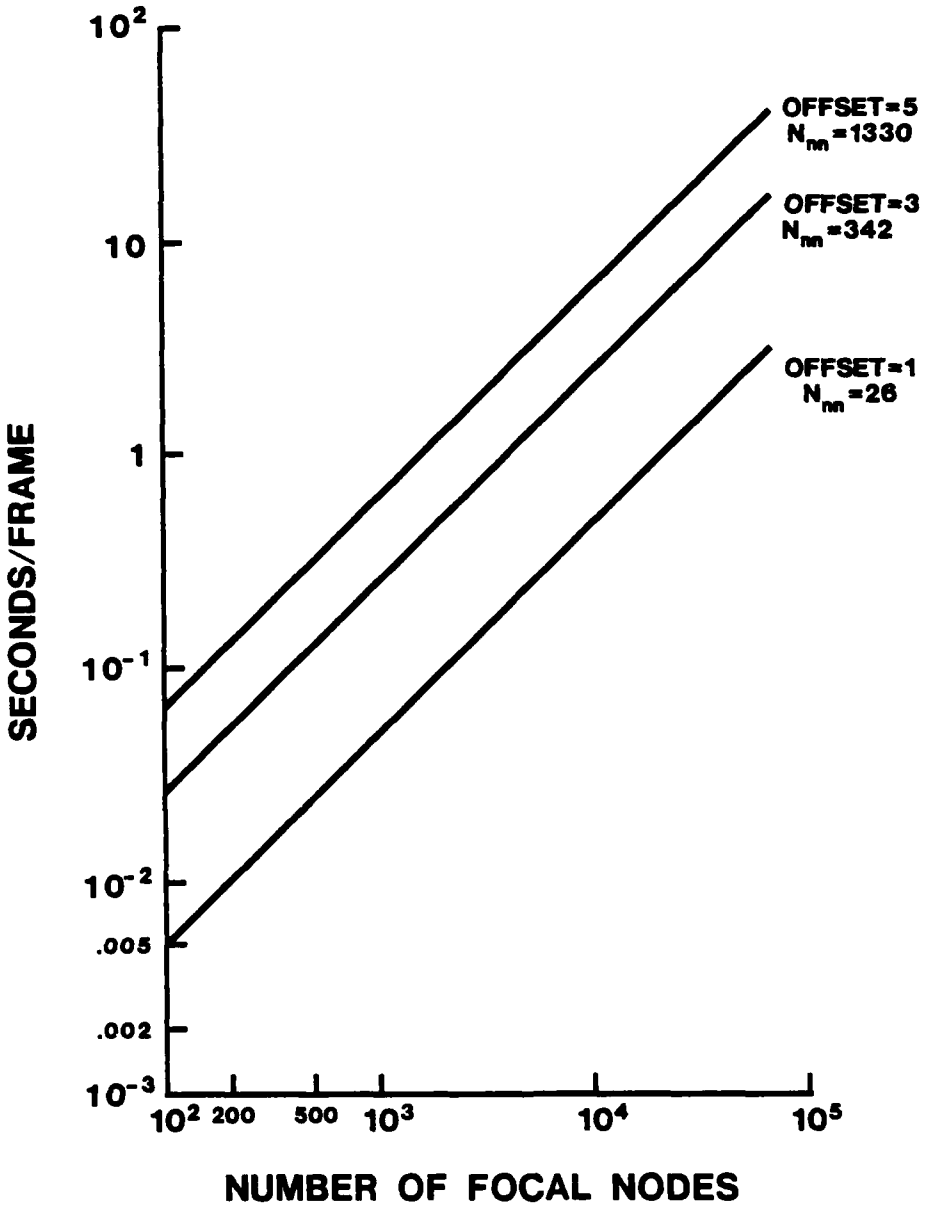


Fig. 2. Cost per frame of using a Monotonic Logical Grid data structure on the NRL Cray X-MP computer. Test 2: Access the data associated with the near neighbors of each focal node and write to a buffer. The total number of nodes is 64 K and the dimensionless time interval between frames is DT = 0.6.

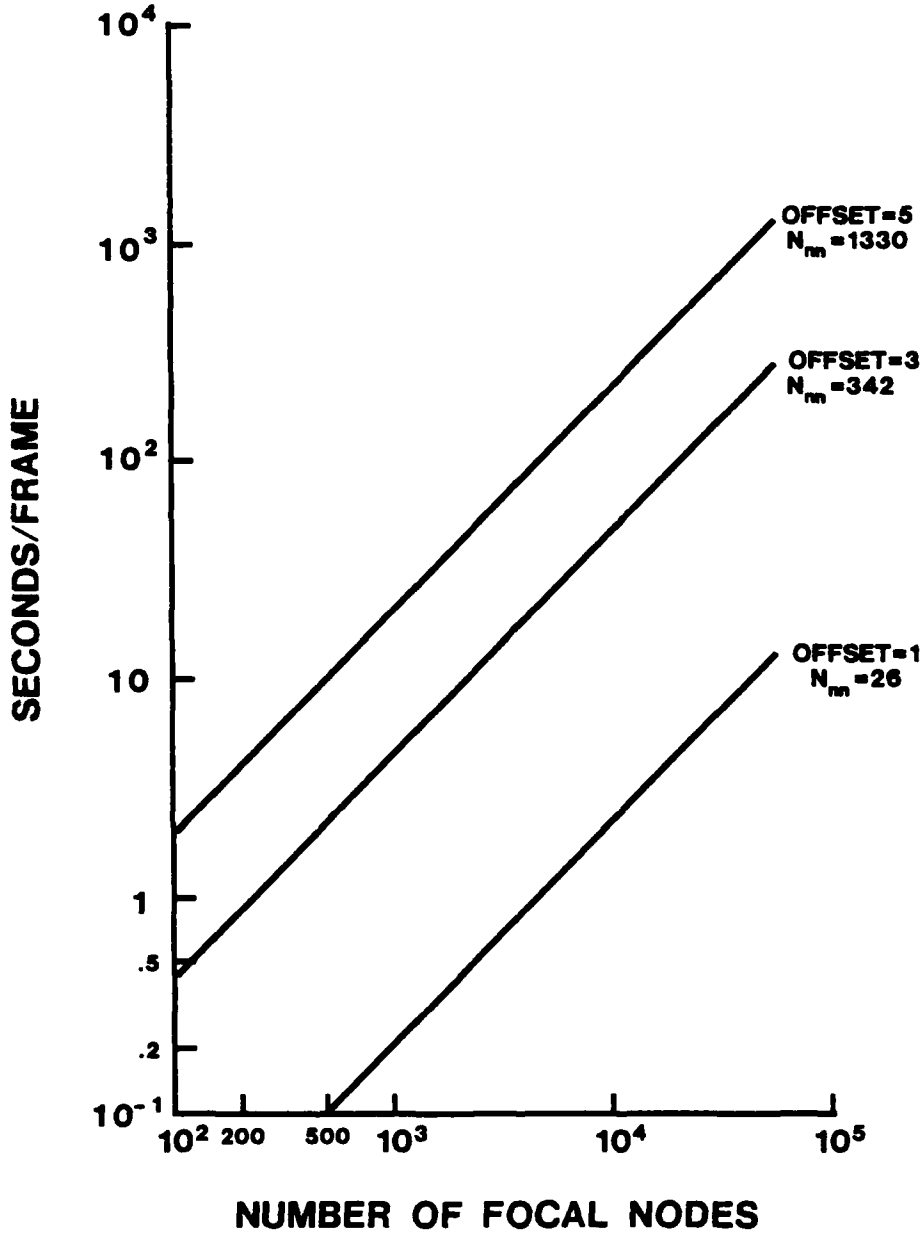


Fig. 3. Cost per frame of using a Monotonic Logical Grid data structure on the NRL Cray X-MP computer. Test 3: Order the near neighbors according to distances from each focal node. The total number of nodes is 64 K and the dimensionless time interval between frames is DT = 0.6.

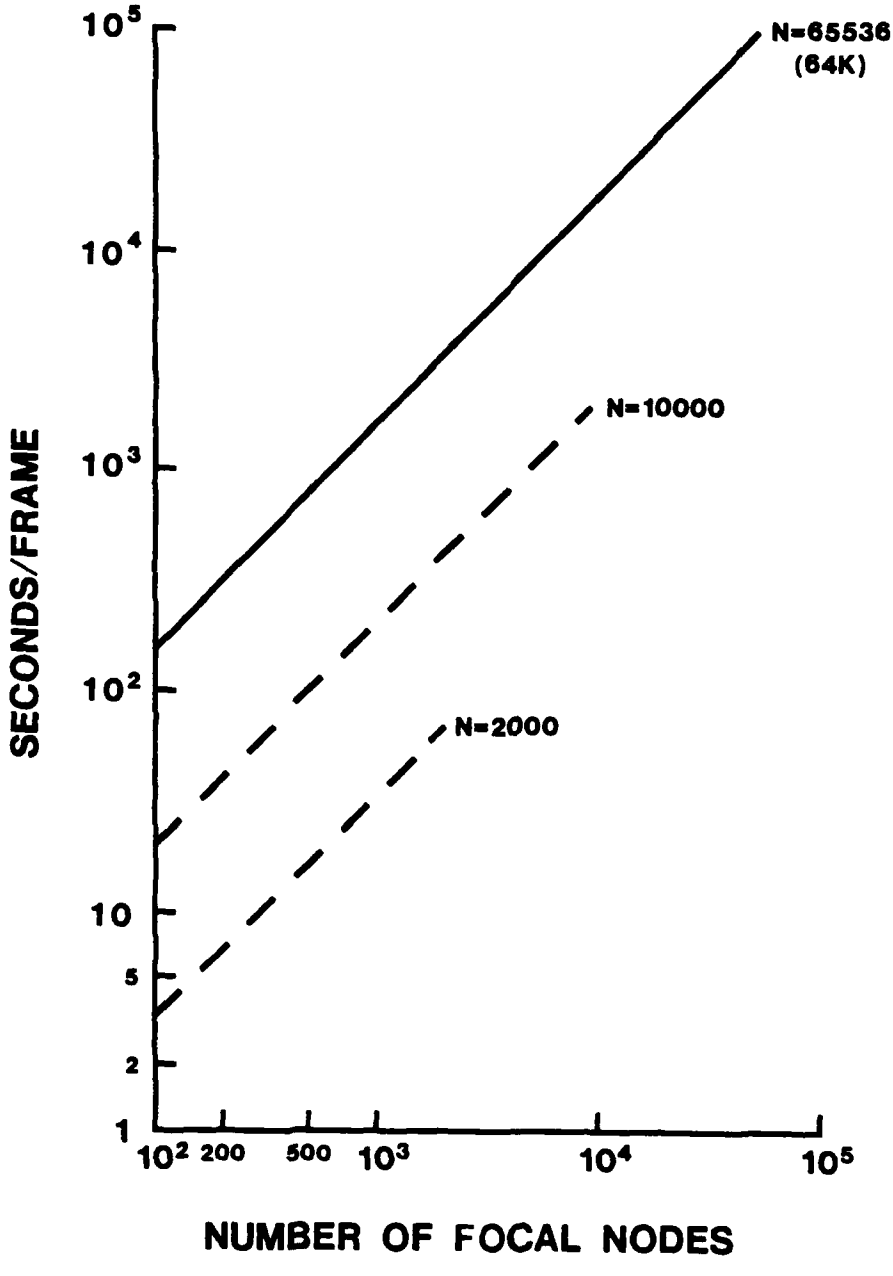


Fig. 4. Cost per frame of using a conventional (Type 1) data structure on the NRL Cray X-MP computer. Test 1: Identify near neighbors of each focal node by using a HEAPSORT routine to order the distances of all nodes from the focal node. The total number of nodes is 64 K. The dashed lines give predicted results for N = 10,000 and 2,000.

spatial density of nodes were high enough that a large fraction of the nodes would influence any focal node. The cost of finding nearest neighbors for $N = 2000$ using the Type 1 data base structure is roughly equal to the MLG cost when $\Delta = 5$, since $N_{nn} = 1330 \sim N$. Experience in molecular dynamics simulations of dimer formation suggests that $\Delta = 4$ is adequate for $N \sim 10^3$ (Lambrakos *et al.*, 1986).

When comparing Figs. 1 and 4, we must remember that the MLG timings are based on $N = 64$K nodes. The value of N determines the MLG restructuring time, which is the minimum value indicated on the graph (≈ 1.6 sec). This restructuring time scales as $N \log N$ and would be much smaller for $N = 2000$, causing the curves to be translated downward on Fig. 1. Thus, going to lower N (dashed curves in Fig. 4) will not give the Type 1 data base structure an edge over the MLG.

Figure 5 shows the cost of the alternative methods of identifying near neighbors using the Type 1 data structure. In the method of Test 1A, the partitioning according to distance from each focal node is much simpler than HEAPSORT. Here we merely ask which nodes are within a prescribed distance of each focal node. This test does *not* provide *nearest* neighbors in contrast to Test 1 above for the Type 1 data structure. *The information content is also much lower than that in the MLG data base.* We used four prescribed distances $R = 2\delta, 3\delta, 4\delta$, and 5δ, as described in Section 2.A. The result scales linearly with the total number of nodes N and the number of focal nodes N_f and is almost identical for all values of R. For Test 1B, we searched for those nodes whose spatial coordinates fell within a prescribed coordinate interval containing a given focal node. We ran the test twice, defining the interval by adding $\pm m\delta$ to each coordinate of the focal node, where m was an integer equal to 2 and 4, respectively. The results were almost identical to those for Test 1A. We may express the cost for Tests 1A and 1B as

$$C_{1A,1B} = N_f \times 0.036 \times (N/65536) \text{ sec/frame.} \tag{5}$$

Comparing Figs. 4 and 5 for the Type 1 data structure, we see that the HEAPSORT (Test 1) costs over an order of magnitude more than testing to see which nodes are within a

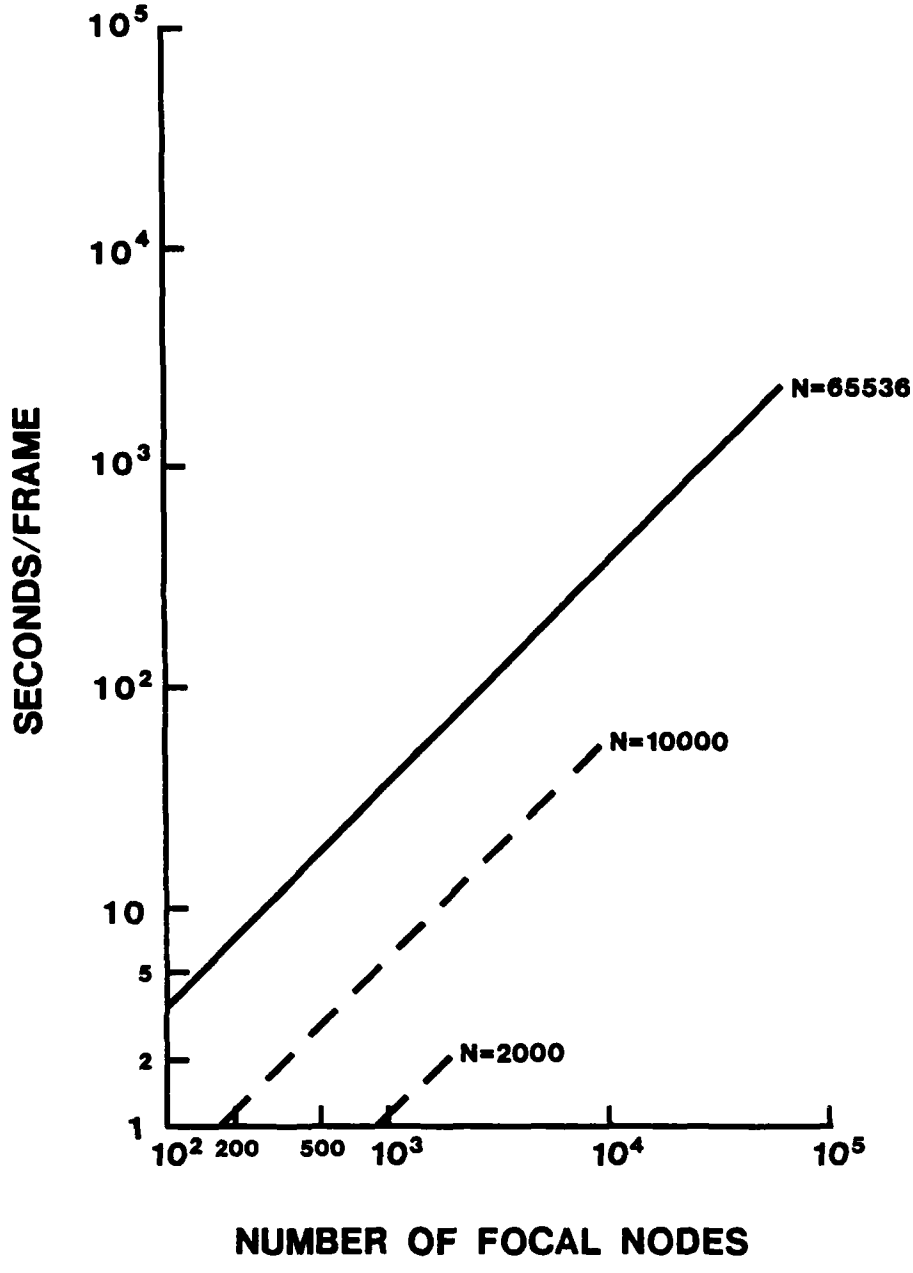


Fig. 5. Cost per frame of using a conventional (Type 1) data structure on the NRL Cray X-MP computer. Test 1A: Identify near neighbors of each focal node by testing to see which nodes fall within a given displacement from the focal node. Results are the same for Test 1B: Identify near neighbors of each focal node by testing to see which nodes have spatial coordinates that fall within a prescribed interval about the coordinates of the focal node. The total number of nodes is 64 K. The dashed lines give predicted results for N = 10,000 and 2,000.

prescribed distance of each focal node or a prescribed coordinate interval (Tests 1A and 1B). *However, the MLG is still faster than the latter method by two orders of magnitude for large numbers of nodes and focal nodes (e.g.,* $N_f \sim N = 64K$*).*

Comparing Figs. 2 and 6, which correspond to the accessing of information on near neighbors, we see that the costs in the two cases are approximately the same. In fact the Type 1 data base costs somewhat less, in apparent contradiction to ideas previously stated. Because the MLG places the near-neighbor data in the vicinity of each focal node, properly designed parallel hardware should be able to access the near-neighbor data for all nodes simultaneously or at least with appreciable parallelism. For a Type 1 data base structure, the indices of near neighbors will be randomly placed in memory and a "gather" operation must be performed. The results of this test point out the importance of the particular hardware. The use of the Cray X-MP is actually optimum for the Type 1 data structure rather than the MLG for two reasons. First, the X-MP has a special hardware gather capability which provides vector speed to the Type 1 random access. Secondly, the Cray vectorizes over only one index, and longer vector loops are more efficient (faster) than shorter ones. The Type 1 data form singly-indexed, long vectors while the data are triply indexed for a three-dimensional MLG. The MLG vector loop is thus necessarily shorter. The results are still comparable because the vector fetches of data in the MLG test are several times faster than the hardware gather instruction, even though the MLG vector loops are short. Ironically the Texas Instruments Advanced Scientific Computer (ASC), which preceded the Cray X-MP at NRL, was a better model for a parallel processor because the ASC could vectorize over more than one index. This would have improved the speed of data retrieval by a factor of three to four when using the MLG data base structure. We point out that the data access times were not large enough to affect the comparisons of the overall costs of the MLG and Type 1 data structures.

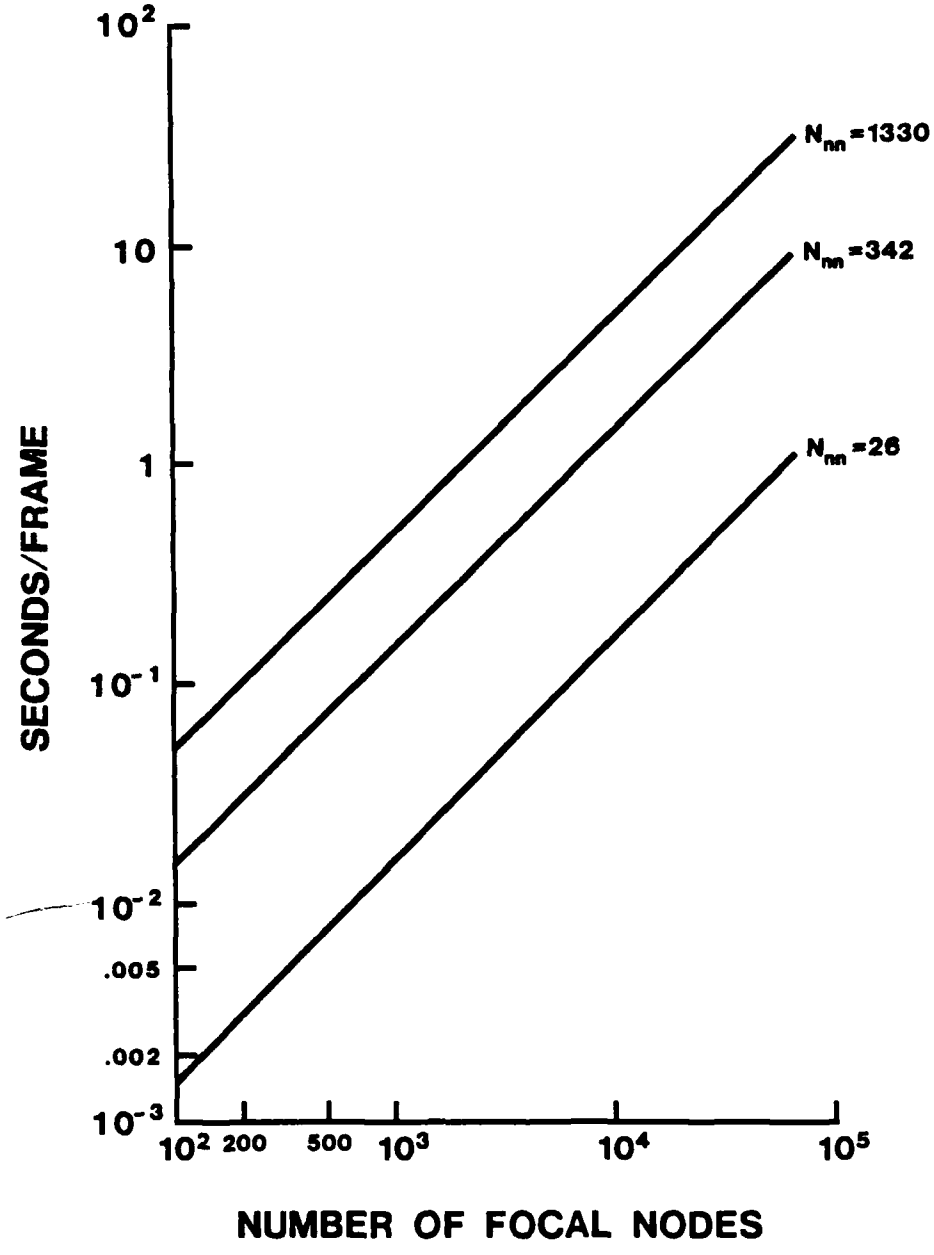


Fig. 6. Cost per frame of using a conventional (Type 1) data structure on the NRL Cray X-MP computer. Test 2: Access the data associated with the near neighbors of each focal node and write to a buffer.

2.B.2. Tables

Tables 1 and 2 show the results of timing measurements of the MLG data base for various values of DT and maximum near-neighbor index offsets from the focal nodes. The sorting times are in units of seconds per frame, since the restructuring of the MLG does not depend on the number of focal nodes being processed. The other timings for the MLG and the timings for the Type 1 data base *do* depend on the number of focal nodes. In both molecular dynamics and target correlation and tracking problems, an appreciable fraction of the nodes will be treated as focal nodes. Thus the first three columns must include a multiplicative factor (N_f) of order 10^4 to obtain the time required per frame to perform each task.

The first three columns of Table 1 correspond to writing information on 27 nodes: the 26 near neighbors of index offset 1 and the focal node itself. This is also the case in Table 3 for the Type 1 data base. Notice that the first three columns of Table 1 do not change with DT while the sorting time increases as DT increases. This is consistent with our statement in Section 2.A that a larger time interval (DT) between frames will cause more swapping to be performed in order to maintain the MLG. The largest proportional jump in sorting time occurs when we cross the DT = 1.0 threshold. Test 3 is of particular interest because the process of ordering the data according to distance is completed. This requires computing the internode distances and then sorting. We can reduce this time further by using a vectorized sorting algorithm of order $N \log N$, so that the timing in Test 3 represents an upper bound.

Table 2 shows the effects of increasing the number of near neighbors accessed per focal node. For an offset Δ, the number N_{nn} of near neighbors is $(2\Delta+1)^3 - 1$. We can see that the cost of writing more data goes up, but not linearly, with N_{nn} for Tests 1 and 2. This is because of the longer vector loops involved in accessing the information and writing to the buffer as N_{nn} increases. For Test 3, the time varies approximately as $N_{nn} \log N_{nn}$, since the HEAPSORT algorithm is not vectorized and has approximate computational

TABLE 1

Times for MLG Data Base Tests with Index Offsets of 1*

DT	Test 1†	Test 2	Test 3	Sorting
0.016	$2.93 \times 10^{-5} N_f$	$5.07 \times 10^{-5} N_f$	$2.05 \times 10^{-4} N_f$	8.88×10^{-1}
0.100	$2.91 \times 10^{-5} N_f$	$5.07 \times 10^{-5} N_f$	$2.05 \times 10^{-4} N_f$	1.10×10^{0}
0.600	$2.91 \times 10^{-5} N_f$	$5.08 \times 10^{-5} N_f$	$2.04 \times 10^{-4} N_f$	1.56×10^{0}
3.000	$2.93 \times 10^{-5} N_f$	$5.07 \times 10^{-5} N_f$	$2.04 \times 10^{-4} N_f$	3.05×10^{0}

*Times are in seconds per frame.

†Excludes the sorting time in column 4.

TABLE 2

Times for MLG Data Base Tests with DT = 0.6*

Δ	N_{nn}	Test 1†	Test 2	Test 3	Sorting
1	26	$2.91 \times 10^{-5} N_f$	$5.08 \times 10^{-5} N_f$	$2.04 \times 10^{-4} N_f$	1.56×10^{0}
2	124	$7.46 \times 10^{-5} N_f$	$1.32 \times 10^{-4} N_f$	$1.33 \times 10^{-3} N_f$	1.53×10^{0}
3	342	$1.44 \times 10^{-4} N_f$	$2.58 \times 10^{-4} N_f$	$4.35 \times 10^{-3} N_f$	1.53×10^{0}
5	1330	$3.49 \times 10^{-4} N_f$	$6.46 \times 10^{-4} N_f$	$2.05 \times 10^{-2} N_f$	1.51×10^{0}

*Times are in seconds per frame. The index offset is the same for MLG indices i, j, and k.

†Excludes the sorting time in column 4.

complexity $M\ log\ M$ for a set of M numbers which must be ordered.

Table 3 shows the results of tests on the the Type 1 data base. Since we have already sorted on distance in Test 1, Test 3 (find *nearest* neighbors) is automatically satisfied. For the MLG to determine a given number of *nearest* neighbors, the index offset Δ must be sufficiently large to include those *nearest* nodes and an additional sort according to distance must be performed. To carry out any calculation which requires the "nearest neighbors", this extra sort is really unnecessary with the MLG structure, provided that the safety factor (a sufficiently large value of Δ) can be tolerated.

Using Table 3, we find that the cost of Test 1 when processing N_f focal nodes for the Type 1 data structure is

$$C_{Test1} = N_f \times 1.6 \times \left[(N\ log\ N)/(65536\ log\ 65536)\right] \text{ sec/frame.} \qquad (6)$$

Equation (6) includes the scaling of HEAPSORT for N nodes. For N = 64 K, Table 2 (column 1 plus column 4) for the MLG and Eq. (6) tell us that the Type 1 data base will require between three and four orders of magnitude more computing time for identifying *near neighbors* when using HEAPSORT than will the MLG data base. If we simply use the method of Test 1A to identify near neighbors, the Type 1 data base structure will still cost two orders of magnitude more for reasonable numbers of focal nodes ($N_f \sim N$). This is because for each focal node, the Type 1 data base must compute 64 K distances and sort them. Thus, the cost of identifying near neighbors scales roughly as $N \times N_f$. For $N_f = N$, cost thus scales approximately as N^2. Maintaining the MLG requires only one global sort, and identifying near neighbors is thus of order N_f.

Table 4 shows how the Type 1 data base structure performs as a function of the number of near neighbors for which data must be accessed. Notice that the ordering of nodes according to distance from each focal node (Test 1) does not change with N_{nn}. However, the cost of accessing the data (Test 2) scales roughly with N_{nn}, as we might expect.

TABLE 3

Times for Type 1 Data Base Tests with $N_{nn} = 26$*

DT	Test 1†	Test 2	Test 3
0.016	$1.56 \times 10^{0} N_f$	$1.66 \times 10^{-5} N_f$	0.0
0.100	$1.57 \times 10^{0} N_f$	$1.63 \times 10^{-5} N_f$	0.0
0.600	$1.58 \times 10^{0} N_f$	$1.63 \times 10^{-5} N_f$	0.0
3.000	$1.57 \times 10^{0} N_f$	$1.62 \times 10^{-5} N_f$	0.0

*Times are in seconds per frame.

†Includes HEAPSORT.

TABLE 4

Times for Type 1 Data Base Tests with DT = 0.6*

N_{nn}	Test 1†	Test 2	Test 3
26	$1.58 \times 10^{0} N_f$	$1.63 \times 10^{-5} N_f$	0.0
124	$1.59 \times 10^{0} N_f$	$5.47 \times 10^{-5} N_f$	0.0
342	$1.57 \times 10^{0} N_f$	$1.40 \times 10^{-4} N_f$	0.0
1330	$1.59 \times 10^{0} N_f$	$5.22 \times 10^{-4} N_f$	0.0

*Times are in seconds per frame.

†Includes HEAPSORT.

Table 5 shows how the cost of identifying near neighbors varies with the number of near neighbors desired in the case of Tests 1A and 1B. Test 1A finds those nodes which are within some prescribed spatial displacement of each focal node while Test 1B finds those nodes whose three spatial coordinates fall within some coordinate interval containing a given focal node. As in the case of Test 1, the timings do not depend on either DT or N_{nn}. In fact, the number of near neighbors which are found in Tests 1A and 1B will depend on the particular configuration of nodes found at a given time and the value of the radius and the coordinate interval, respectively, that have been chosen to define which nodes are "near" neighbors.

TABLE 5

Times for Alternative Type 1 Data Base Tests with DT = 0.6*

N_{nn}	Test 1A†	Test 1B‡
30-70	$3.63 \times 10^{-2} N_f$	$3.57 \times 10^{-2} N_f$
100-300	$3.63 \times 10^{-2} N_f$	
500	$3.64 \times 10^{-2} N_f$	$3.79 \times 10^{-2} N_f$

*Times are in seconds per frame.

† Find nodes within a prescribed distance of a given focal node.

‡ Find nodes with coordinates near those of a given focal node.

3. Monotonic Logical Grid vs. "Type 2" Data Base Structures

Here we will briefly describe another class of data structures applicable to collections of moving nodes with interactions between near neighbors. These data structures, which we denote by "Type 2" (T2), have a computational complexity of order N for computing interactions, as does the MLG. However, unlike the MLG, the T2 structures require extra memory to carry information in the form of pointers that indicate which nodes are near neighbors. This makes the T2 structures significantly more complex than the MLG, which simply rearranges memory (index space) to reflect near-neighbor information. The T2 pointers often appear in the form of linked-list variables that relate spatial groupings of particles. This feature could render them unvectorizable. As an example, we will describe the method of Hockney and Eastwood (1981), which we denote by "HET2". Another T2 method is that of Appel (1985), which uses a hierarchy of clusters to represent near-neighbor relationships. We point out that both of these papers emphasize the methods of computing Coulombic or gravitational near-neighbor interactions as much as the actual construction of the data structures.

The HET2 data structure covers the physical space with a set of identical cubical cells, the length of a side being determined by the range of the near-neighbors interaction. Each cell has a "head of chain" (HOC) variable which is one word in length, and each moving node has a linked-list (LL) variable in addition to the other associated data (e.g., position, velocity). The associated data are indexed by the particle ID as in the Type 1 data structure. The HOC and LL variables form chains which identify all of the node IDs associated with each spatial cell. In this way, the scheme maintains coarse information on near neighbors: the near neighbors of a node are the other nodes in the same cell plus perhaps those in the adjoining cells.

We use a simple example to illustrate how this is done. Suppose the number of nodes N is 100 and that at time t the nodes with ID = 10, 30, and 50 lie in cell number 12. At each time step or frame, the algorithm cycles through the position data of all of the

nodes in order of ID value. Before the loop through node IDs is started, all cells have an HOC of 0. For each successive node, beginning with ID = 1, three divide operations are performed to determine which cell contains the node. When we get to node ID = 10, we find that node 10 is in cell number 12. We set LL(10) = 0, the value of HOC for cell 12 (that is, HOC(12)). We then set HOC(12) = 10, the ID of the first node found to lie in cell 12. When we get to node number (ID =) 30, we again find that this node is in cell 12. We set LL(30) = HOC(12) = 10, the ID of the first node which we found. We then set HOC(12) = 30, the ID of the second node which we found to lie in cell 12. When we arrive at ID = 50, we find the third node occupying cell 12. We set LL(50) = HOC(12) = 30 and HOC(12) = 50. After the loop over IDs is finished, we may find which nodes are in cell 12 by reading HOC(12) to obtain ID = 50, then reading LL(50) to obtain ID = 30, and finally reading LL(30) to obtain ID = 10. Because LL(10) = 0, we know that only those three nodes lie in cell 12.

After this procedure is performed, the "near neighbors" of a given node are then all nodes found in the same cell plus the nodes in some set of the *nearest* neighboring cells. As in the case of the MLG, this is only coarse information on relative location, and further distance calculations and sorting may be required. The details of updating or recomputing the linked lists at each time step will vary with user and application.

We need not perform test calculations to observe several advantages of the MLG over this T2 class of schemes:

(1) Because the MLG manipulates computer memory directly rather than external mapping of the memory (HOC and LL variables), the use of memory is more efficient. The HET2 scheme requires extra memory in the form of one HOC per cell and one LL variable per node, and an appreciable number of the cells could be empty for cases with many nodes whose trajectories are not random.

(2) Depending on the problem, the optimum sizes and and numbers of cells might vary significantly over time, rendering the HET2 scheme impractical.

(3) The above HET2 structure is not particularly vectorizable and cannot take advantage of a hardware gather capability like that on the NRL Cray X-MP. We can replace the linked-list concept by separate list vectors for each cell to take advantage of the hardware gather capability. This would drastically increase the requirements for memory or at least the difficulty in allocating memory efficiently.

(4) Local restructuring of the linked-lists from frame to frame is not vectorizable in the above prescription and may not permit vectorization in general.

(5) The T2 algorithm is more complex and cumbersome than the MLG algorithm and thus more susceptible to programming errors.

(6) Extention of T2 to four-dimensional (space-time) correlation problems or even higher dimensions is far less transparent than for the MLG.

The extension of the MLG data base structure to higher dimensions could be quite important in some calculations. Attributes distinguishing nodes from each other would in fact determine how various nodes participate in the calculation. For example, in a data correlation problem, suppose that the nodes represent detections of moving vehicles by an array of sensors. In a determination of which data came from the same target, the characteristics of signals produced by different types of vehicles could provide a basis for eliminating a significant fraction of sensor data from consideration. This would greatly improve the speed of the calculation. The MLG data base structure could include extra dimensions corresponding to various vehicle attributes, so that near neighbors in index space would be geographic near neighbors *with the same attributes.* One can envision using a number relational operators in addition to "less than or equal to" ($\leq$), which has been used to construct the 3D MLG data base structure considered in this paper.

4. Conclusions Regarding Target Tracking and Correlation

We have investigated the relationships of the MLG data base structure to two standard data base structures and compared the different algorithms for several tasks:

(1) Identifying the near neighbors of a set of focal nodes within a set of noninteracting nodes that move randomly,

(2) Retrieving information on the near neighbors that were identified in (1), and

(3) Ordering the information retrieved on the near neighbors. The order corresponded to the distances of near neighbors from the respective focal nodes.

In target tracking and correlation calculations, a large fraction of the total set of nodes will have to be treated as focal nodes throughout the evolution of the system. Thus the access of data on near neighbors will scale as N^2 – the so-called "combinatorial explosion" – for most conventional data bases. Such conventional structures generally fall in the Type 1 category considered above. Even if we use the more clever Type 2 approaches considered in Section 3, the implementation and performance will fall short of the MLG. In the latter case, the source of the advantage is that the MLG restructures data in computer memory directly – without a cumbersome external apparatus like linked lists.

We have also identified two parameters which will determine the performance of the MLG: the dimensionless frame time, DT, and the index offset, Δ. For the case of large numbers of projectiles traveling together at similar speeds and trajectories, the relative velocities of neighboring nodes will be much smaller than the absolute velocities, unlike the test cases reported here. Thus the values of DT will be less than 1 and the MLG will be easy to maintain. While the required index offset will depend on the density of nodes and the desired number of *nearest* neighbors, we would be surprised if offsets larger than 5 would be necessary.

Our findings thus support the previous assertion that the MLG provides powerful advantages for the surveillance, tracking, and correlation problem. We have also identified

major sources of strength of the MLG approach: the requirement for only one reordering process per frame, the order N scaling of calculations of near-neighbor interactions and associations, the ready implementation on vector or highly parallel computers, and the direct restructuring of memory, eliminating the need for an expensive, partly scalar apparatus (e.g., linked lists) for the maintenance of near-neighbor identification data. As an interesting side issue, we have found that the computer hardware can directly impact the comparisons of data base structures. In this case, the Cray X-MP hardware gather capability permitted the Type 1 data base to retrieve near-neighbor data somewhat more quickly than for the MLG data base, when both were programmed in FORTRAN. This was not, however, a significant factor in the comparison.

5. Acknowledgements

The authors gratefully acknowledge support provided by the Office of Naval Research and the Strategic Defense Initiative Organization. We also appreciate many helpful discussions with Mr. M. S. McBurnett of the NRL Information Technology Division.

Appendix

Constructing a Monotonic Logical Grid – An Example

Begin with a collection of 16 objects randomly distributed in space as shown in Fig. A1.

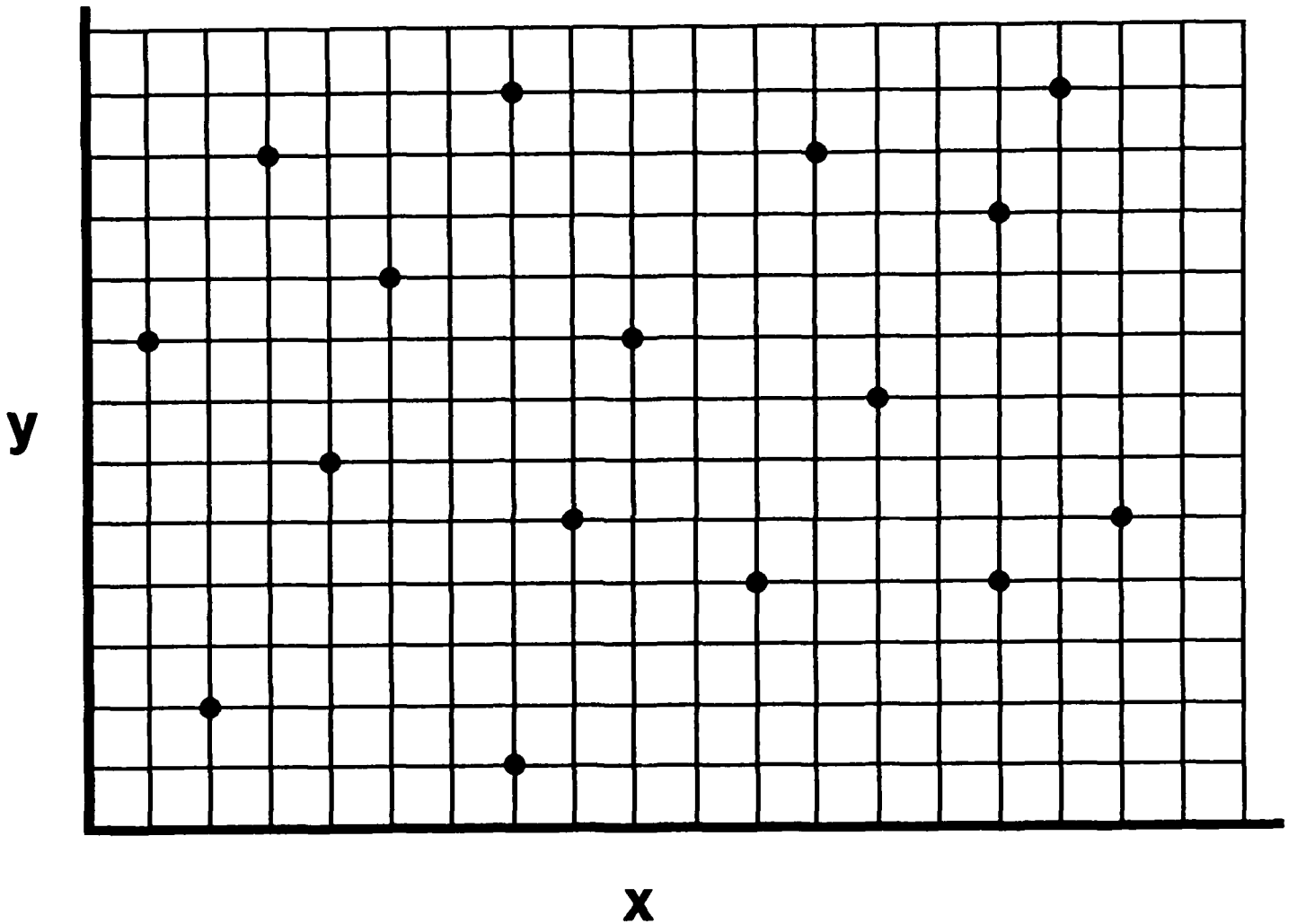


Figure A1

We will organize these in a section of computer memory so that near neighbors in space will be near-neighbors in index space. This indexing scheme or rule for the data organization in memory is an example of a monotonic logical grid (MLG).

Step 1. Give each object x and y coordinates in 2D space relative to some origin. This has been done in Fig. A1.

Step 2. Decide on the "type" of MLG one wants. This decision is expressed in terms of a rule for constructing the grid. The rule which we use may be stated as follows:

Rule. Choose a computer memory location or index (i, j) for each object such that

$$x(i, j) \leq x(i+1, j)$$

$$y(i, j) \leq y(i, j+1)$$

where i and j run from 1 to 4.

Step 3. Implement the rule as follows, remembering that the MLG which we have chosen will occupy a 4×4 index space in memory:

A) Order all sixteen objects according to increasing y-coordinate, that is, from lowest to highest y-value. In Fig. A2 below, we have lettered these from A to P.

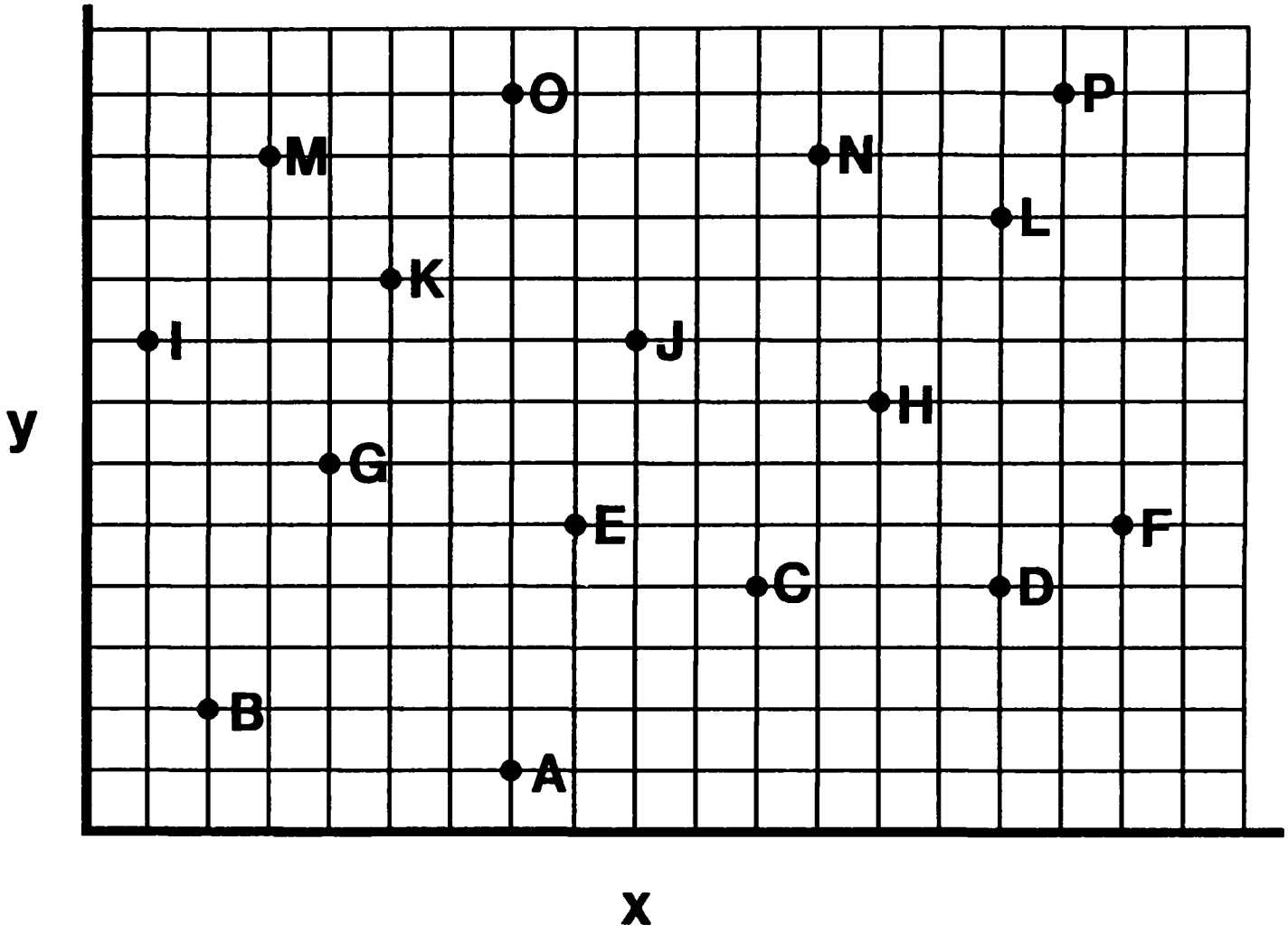


Figure A2

B) Give the objects with the four lowest y-values (that is, objects A, B, C, and D) a j index of 1. Give those with the next four lowest y-values (E, F, G, H) a j-index of 2 and so on. We can connect each set of four with straight lines as shown in Fig. A3.

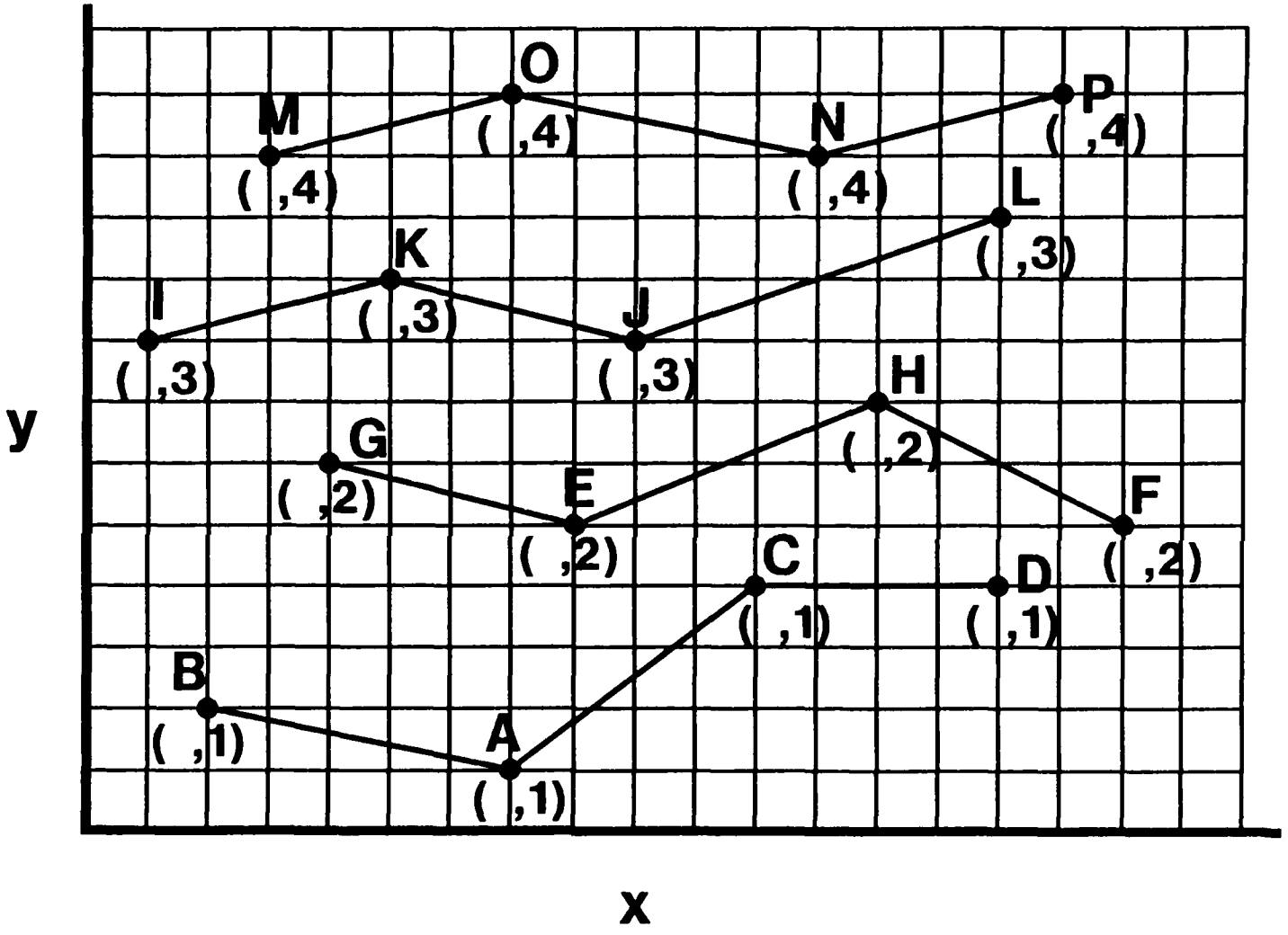


Figure A3

C) Now assign an i-index to each object in a given set of four with the same j-index according to the selected MLG rule. This assignment is in Fig. A4. Note that we have again connected those objects with the same i-index with a line.

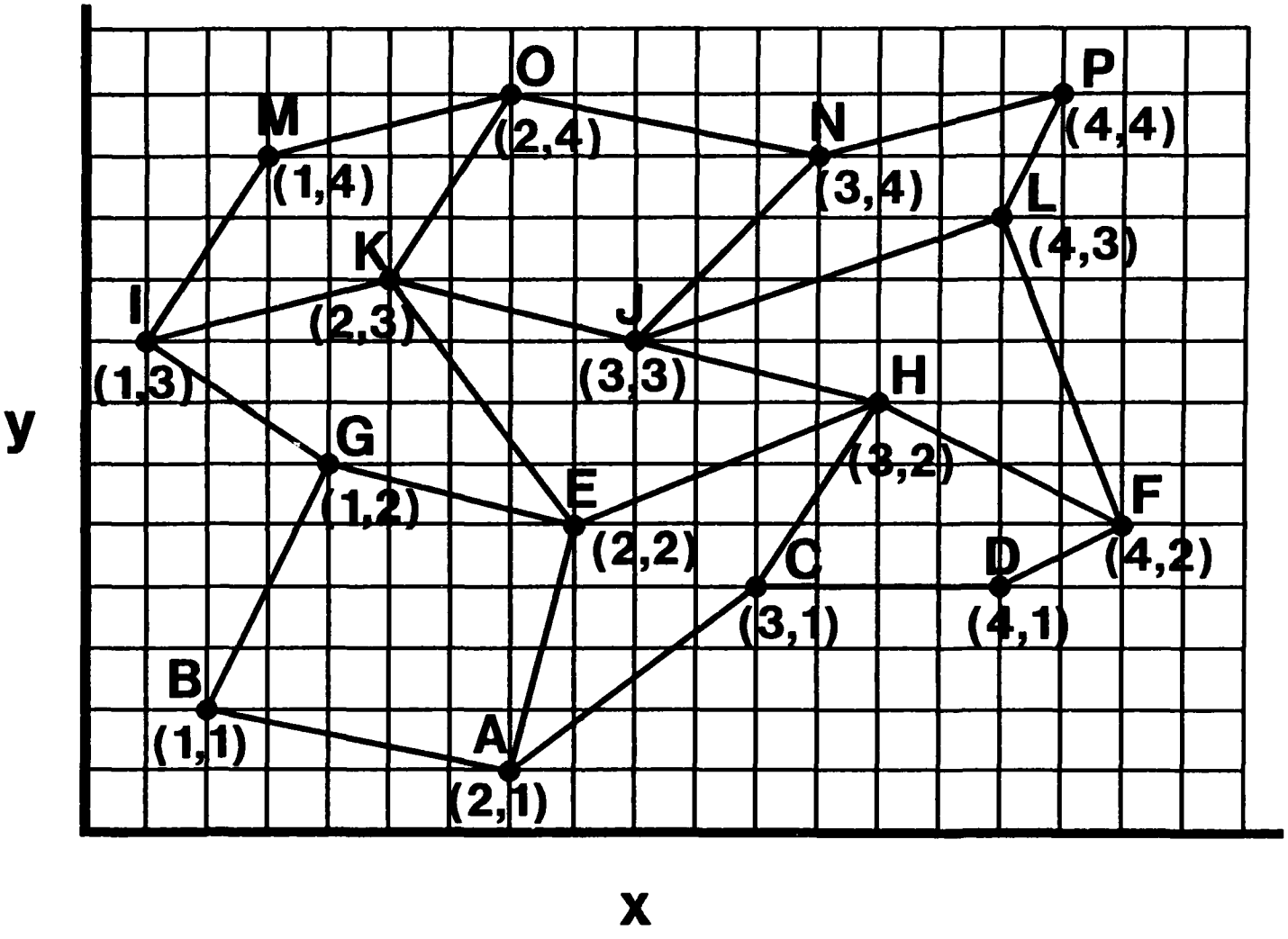

O
P
M
(2,4)
N
(4,4)
(1,4)
(3,4)
L
K
(4,3)
I
(2,3)
J
(1,3)
(3,3)
H
y
G
(3,2)
(1,2)
E
F
(2,2)
C
D
(4,2)
(3,1)
(4,1)
B
(1,1)
A
(2,1)
x

Figure A4

The 2D MLG in index space looks like Fig. A5. The regular grid depicted for memory (index space) has lines which correspond directly to the lines which we drew in Figs. A3 and A4 when constructing the MLG indices.

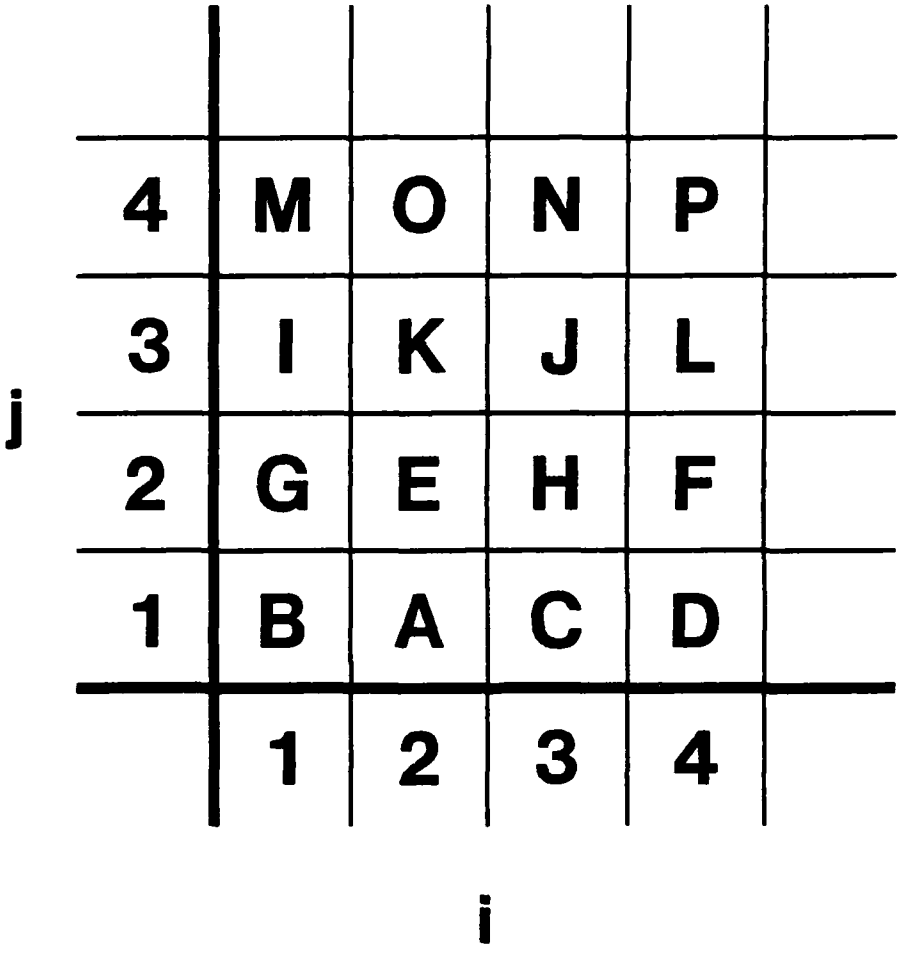


Figure A5

The eight near neighbors of point H in index space are points A, C, D, E. F, K. J, and L in real space, as we can see in Fig. A6. The dashed line in Fig. A6 encloses the near neighbors of H in index space (compare with Fig. A5).

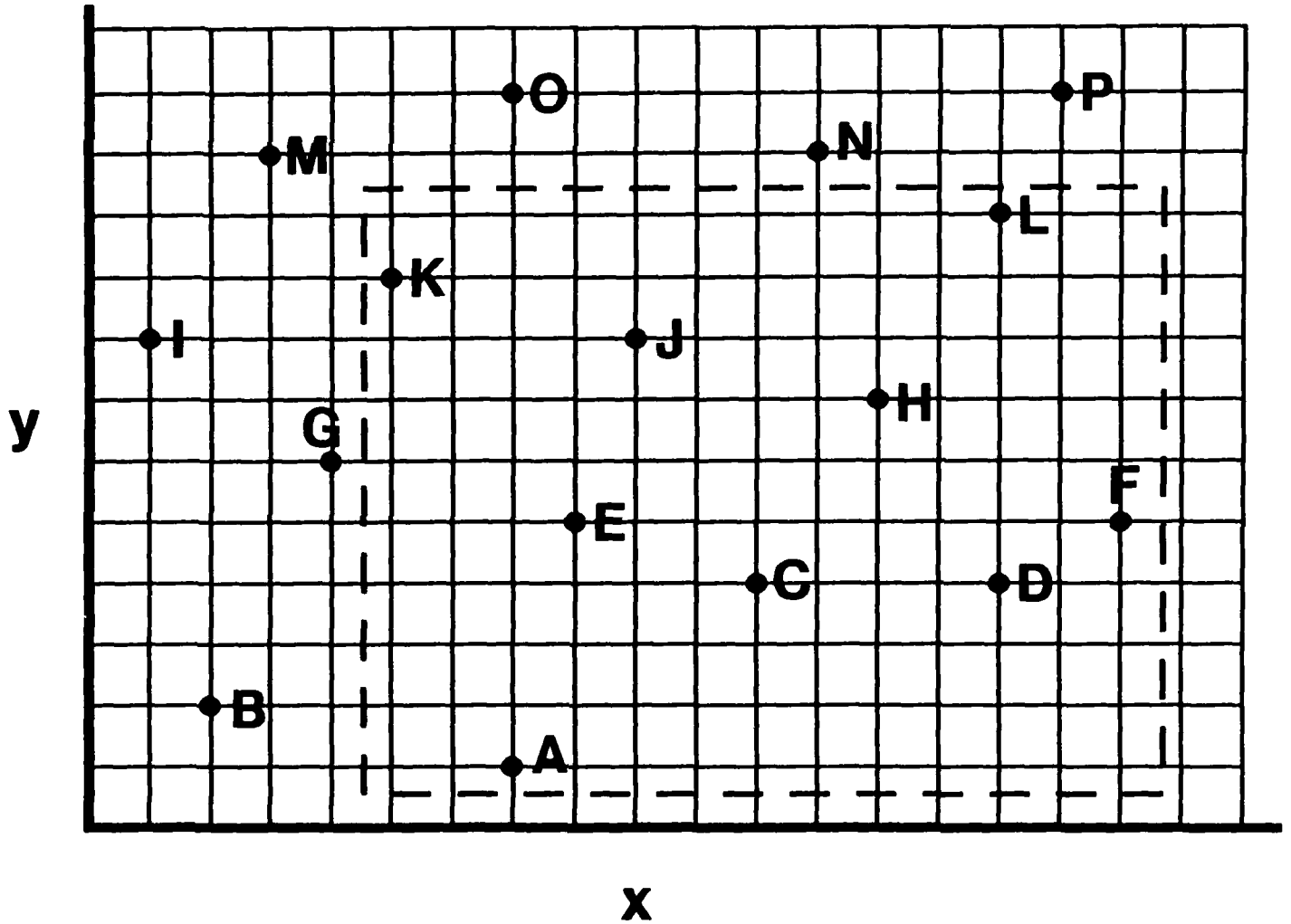


Figure A6

References

Appel, A. W. (1985) An Efficient Program for Many-Body Simulation, SIAM J. Sci. Stat. Comput. **6**, 85.

Boris, J. P., Picone, J. M., and Lambrakos, S. G. (1986) Beast: A High-Performance Battle Engagement Area Simulator/Tracker, NRL Memorandum Report, to be published.

Hockney, R. W. and Eastwood, J. W. (1981) **Computer Simulation Using Particles**, McGraw-Hill, New York, chapter 8, pp. 267-309.

Lambrakos, S. G. and Boris, J. P. (1986) Geometric Properties of the Monotonic Logical Grid Algorithm for Near-Neighbor Calculations, J. Comp. Phys., to be published.

Lambrakos, S. G., Boris, J. P., Guirguis, R., Page, M., and Oran, E. S. (1986) Molecular Dynamics Simulation of Dimer Formation in a System of Asymmetric Molecules, J. Chem. Phys., to be submitted.

Nijenhuis, A. and Wilf, H. S. (1978) **Combinatorial Algorithms for Computers and Calculators**, Academic Press, New York, 140.

Reid, Donald B. (1979) An Algorithm for Tracking Multiple Targets, IEEE Transactions on Automatic Control, Vol. AC-24, 843.

END

12-86

DTIC